Commanding Constellations (Pipeline Architecture)

Tim Ray, Jeff Condron
NASA/GSFC
Timothy.J.Ray@nasa.gov

Abstract

Providing ground command software for constellations of spacecraft is a challenging problem. Reliable command delivery requires a feedback loop; for a constellation there will likely be an independent feedback loop for each constellation member. Each command must be sent via the proper Ground Station, which may change from one contact to the next (and may be different for different members). Dynamic configuration of the ground command software is usually required (e.g. directives to configure each member's feedback loop and assign the appropriate Ground Station). For testing purposes, there must be a way to insert command data at any level in the protocol stack.

The Pipeline architecture described in this paper can support all these capabilities with a sequence of software modules (the pipeline), and a single self-identifying message format (for all types of command data and configuration directives). The Pipeline architecture is quite simple, yet it can solve some complex problems. The resulting solutions are conceptually simple, and therefore, reliable. They are also modular, and therefore, easy to distribute and extend. We first used the Pipeline architecture to design a CCSDS (Consultative Committee for Space Data Systems) Ground Telecommand system (to command one spacecraft at a time with a fixed Ground Station interface). This pipeline was later extended to include gateways to any of several Ground Stations. The resulting pipeline was then extended to handle a small constellation of spacecraft. The use of the Pipeline architecture allowed us to easily handle the increasing complexity.

This paper will describe the Pipeline architecture, show how it was used to solve each of the above commanding situations, and how it can easily be extended to handle larger constellations.

1. The Pipeline Concept

At its simplest, a pipeline consists of a serial chain of software modules. For example, a pipeline to implement a protocol stack might consist of one module for each protocol layer. Inputs to the pipeline are in the form of messages, which enter the pipeline at the front end, and make their way serially through the pipeline. A generic message format is used, and every module in the pipeline uses the generic format for both input and output of messages. The exact message format chosen is not critical, except that the messages must be self-identifying (e.g. each directive must be wrapped with a label indicating the *type* of content, such as "directive for the xyz protocol layer").

Again, messages always enter the pipeline at its front end. If the message is a directive for the first module, the first module will "eat" the message and execute the directive. If the first module does not recognize the message type, it will forward the message to the second module. In this way, all messages automatically make their way serially through the pipeline until they reach their intended destination. It is also possible that a module will eat one message and insert another. For example, a module that implements the framing layer of a protocol stack may eat all *Packet-Data* messages and replace them with *Frame-Data* messages.

2. First pipeline: Build spacecraft commands

The pipeline concept was conceived during informal discussions about how to provide a generic CCSDS commanding system (i.e. capable of commanding any CCSDS-compliant spacecraft). To test the pipeline concept, we implemented the ground (sending) end of the CCSDS Telecommand protocols. These protocols provide reliable delivery of command data to a spacecraft. This first pipeline contained these modules:

Framing (puts Packets into Frames)
Coding (puts Frames into Codeblocks)
Synch (puts Codeblocks into Transmission-Units)

2.1. Flow of command data

The source of spacecraft commands builds CCSDS command Packets; we call this the **Message-Source**. It puts each Packet into a Packet message, and sends the message to the pipeline. Consider what happens when a Packet message enters the pipeline. First, remember that *all* messages enter the pipeline at the front end (for this pipeline, the Framing module). The Framing module recognizes a Packet message, and eats it. The Packet from inside the message is used to build a Frame, and this Frame is then passed to the Coding module (as a Frame message). The Coding module recognizes the Frame message, and eats it. The Frame from inside the message is used to build Codeblocks, and these Codeblocks are then passed to the Synch module (as a Codeblocks message). The Synch module recognizes the Codeblocks message, and eats it. The Codeblocks from inside the message are used to build a Transmission-Unit, which is passed out of the pipeline (as a Transmission-Unit message).

2.2. Flow of directives

Consider what happens when a directive for the Framing module enters the pipeline. The Framing module recognizes the Framing-Directive message, and eats it. The directive from inside the message is executed.

Now consider what happens when a directive for the Synch module enters the pipeline. The Framing module does not recognize the Synch-Directive message, so the message is passed through to the Coding module. The Coding module does not recognize the Synch-Directive message either, so it also passes the message on (to the Synch module). The Synch module recognizes the Synch-Directive message and eats it. The directive from inside the message is executed.

2.3. Hierarchical messages

As mentioned earlier, we use a hierarchical message format in our pipelines. The format specifies an inner container that we call an **envelope** (think of a letter to be mailed) and an outer container that we call a **mailbag**. Each "message" is actually a mailbag that may contain multiple envelopes. The hierarchical message format provides more power and flexibility, but does not affect the pipeline concept. Each module reacts to the parts of a message that it recognizes, and passes any unrecognized parts on to the next module in the pipeline.

3. Second pipeline: Add Ground Stations

Our first pipeline provided the generic command-building capabilities that we wanted, and was easy to implement and operate. However, in order to be useful, the Transmission-Units need to be delivered to the spacecraft. This required adding a layer to our pipeline, which we called the **Gateway** layer. To operate a satellite, the Gateway must send the Transmission-Units to a Ground Station, which has the antennas that transmit data to the spacecraft. In order to maintain our philosophy of providing generic solutions, we needed to provide an interface to *all* NASA Ground Stations. There are three families of NASA Ground Stations: the Ground Network (GN), Space Network (SN), and Deep Space Network (DSN). Each family has a unique interface, so we added three different Gateway modules. This is the resulting pipeline:

```
    Framing
    Coding
    Synch
    Gateway_GN   Gateway_SN   Gateway_DSN
```

3.1. Choosing which Gateway module to use

Note that this pipeline is not entirely serial in nature – there are 3 possible routes through the Gateway layer. However, the pipeline will appear to be serial – only one of the Gateways will be used at a time; i.e. each message will enter the pipeline at the Framing module, then go through the Coding module, Synch module, and one of the Gateway modules.

All modules include a directive to choose "which module is next". This allows the Message-Source to decide which Gateway module to use (by sending a Synch-Directive envelope containing a "which module is next" directive). If desired, the Message-Source can change the configuration on-the-fly.

3.2. Configuring the chosen Gateway module

Once a Gateway module is chosen (i.e. becomes part of the current pipeline topology), it can be configured. For example, suppose the Gateway_DSN module is chosen. The Message-Source sends messages containing Gateway_DSN-Directive envelopes to the pipeline. None of the original pipeline modules recognize the Gateway_DSN-Directive envelope, so these envelopes are passed through the pipeline until they reach the Gateway_DSN module, where they are recognized, eaten, and executed.

3.3. Flow of command data

Command data flows through the original pipeline modules just as before; the result is always a Transmission-Unit envelope. This Transmission-Unit envelope is passed on to the current Gateway module, which recognizes it, and eats it. The Transmission-Unit from inside the envelope is wrapped in the appropriate header (as required by each Ground Station family), and sent to the spacecraft via the Ground Station.

3.4. No changes to existing modules & interfaces

This extension to the original pipeline did not require any changes to the existing modules, nor were any changes made to the interface between the Message-Source and the pipeline (i.e. the message format was not changed).

4. Third pipeline: Small constellation

The multi-mission pipeline was further extended to support commanding of a small constellation (3 spacecraft). This constellation uses 3 copies of the Framing module, because each spacecraft requires an independent feedback loop for reliable delivery of Frames. A Framing-Routing layer is needed in front of the Framing layer, so that each incoming message can be routed to the appropriate Framing module. One copy of the Coding and Synch modules is sufficient, because all 3 spacecraft use the same setup for these protocol layers. (and operational constraints ensure that a Transmission-Unit will never contain commands for more than one spacecraft). This pipeline contains these modules:

Framing-Router
Framing_1 Framing_2 Framing_3
Coding
Synch
Gateway_GN Gateway_SN Gateway_DSN

As with the previous pipeline, although the pipeline is not entirely serial, any particular message takes a serial path (visits one module at each layer).

4.1. Framing-Router module

The hierarchical message format is beneficial here. For this mission, each mailbag from the Message-Source includes an envelope with the label "Spacecraft-ID", along with the usual data or directive envelope. This Spacecraft-ID envelope is recognized by the Framing-Router module, which uses it to route each incoming message to the appropriate Framing module.

4.2. Flow of command data

The Message-Source sends messages that contain both a Packet envelope and a Spacecraft-ID envelope. Each message enters the pipeline at the front end (in this case, the Framing-Router module). The Framing-Router module doesn't recognize the Packet envelope, so it leaves this envelope in the message. It does recognize the Spacecraft-ID envelope, and uses it to decide which module is next in the pipeline (i.e. which of the Framing modules). Once the Packet envelope reaches a Framing module, it flows through the remainder of the modules in the usual way. The result is that a Transmission-Unit is sent to the spacecraft via the chosen Gateway module.

4.3. No changes to existing modules & interfaces

As with the first extension of the pipeline, this extension did not require any changes to the existing modules. Nor were any changes made to the interface between the pipeline and the Message-Source (the message format was not changed).

4.4. Multi-mission support

This pipeline is currently being used to operate three different NASA missions. With the addition of one more Gateway module (to send data to the spacecraft via an RS422 card) the pipeline became capable of supporting spacecraft integration & test. The resulting Command system has been, is, or will be used in 25 different NASA facilities spanning 5 different missions. The system is used throughout the mission lifetime (flight hardware/software development, spacecraft integration & test, spacecraft operations, and flight software maintenance).

Each facility uses the same Message-Source and the same pipeline (an identical set of programs built from the same source code). The Message-Source dynamically configures the pipeline to match each spacecraft's setup (e.g. Spacecraft-ID, maximum Frame size, which Gateway to use, etc).

5. Commanding larger constellations

The commanding of larger constellations can easily be handled by the pipeline architecture. Let's consider some additional extensions to the existing pipeline.

As the constellation grows in size, additional Framing modules must be added (so that each spacecraft has an independent feedback loop for reliable delivery of commands). These modules would be copies of the generic Framing module, and be configured in the same way as those in our existing pipeline. The Framing-Router module would continue to choose which Framing module to use for each incoming message.

If the constellation requires the use of several Ground Stations concurrently, a Gateway-Router layer can be inserted in front of the Gateway layer. This layer would be similar to the Framing-Router layer – it would use the "Spacecraft-ID" envelope to decide which of the available Gateways to route the message to. As in the Framing-Router, there would be a table that specifies a route (i.e. module name) for each possible Spacecraft-ID value. Dynamic configuration of the Gateway-Router could be accommodated with a Gateway-Router directive – the directive would modify one entry in the table (e.g. "Spacecraft-ID 23 will now use the Gateway_GN module").

As the pipeline grows, there may be a need to distribute it across multiple machines. This is easily accommodated, and is discussed in the next section.

6. Summary

As we have seen, a pipeline can easily be extended, without changing any of the existing modules, nor the interface between the pipeline and the Message-Source. One type of extension involves adding modules to the front, middle, or end of the pipeline. The use of one generic message format for *all* module interfaces makes it is easy to insert or delete modules as needed. Another type of extension involves expanding a particular layer (e.g. the multiple Framing-layer modules needed by our small constellation). This is easily handled through the use of multiple copies of the layer's module, and the insertion of a Router module. The Router module chooses which of the available modules to use.

Pipelines can easily be dynamically configured. The topology of the pipeline can be modified on-the-fly by sending directives to each module specifying "which module is next". Each module within the pipeline can then be configured individually by sending directives. Changes in the pipeline's topology do not require any changes to the Message-Source – its only interface is to the front end of the pipeline.

A pipeline can easily be distributed across multiple machines if desired. Changes in how the modules are distributed do not require any changes to the Message-Source – again, its only interface is to the front end of the pipeline.

The pipeline concept is not tied to any particular programming language; pipelines can easily be implemented in C, C++, C#, Java, Python, etc.

Although the Pipeline Architecture is simple in concept, it is powerful. Our Telecommand pipeline supports both individual spacecraft and constellations, and is working reliably in many NASA facilities.